# USING VISUAL PROGRAMMING TO SIMULATE, TEST, AND DISPLAY A TELEMETRY STREAM "

George Wells and Ed Baroth, Ph.D.
Measurement Technology Center,
Jet Propulsion I dm-story,
California institute of '1'ethnology

## ABSTRACT

Advantages of using visual programming to create, modify, test and display a telemetry stream are presented. Commercial visual programming software is being used to test new algorithms as part of the ground support for the Galileo spacecraft Test Bed. It is very important that any new software algorithms be thoroughly tested on the ground before any modifications are made to the spacecraft.

Visual programming provides easy visibility into the decommutation process, including real-time data display and error detection. A data acquisition board is used to clock in the actual synchronous telemetry signal from the Test Bed at rates below 10 k? Iz. Time to write and modify code using visual programming is significantly less, by a factor of 4 to 10, than using text-based code. The gains in productivity are attributed to the communication among the customer, developer, and computer that are facilitated by the visual syntax of the language.

## INTRODUCTION

The Measurement Technology Center (MT(') evaluates commercial data acquisition, analysis, display and control hardware and software products that are then made available to experimenters at the Jet Propulsion Laboratory. In addition, the MTC acts as a systems integrator to deliver turn-key measurement systems that include software, user interface, sensors (e. g., thermocouples, pressure transducers) and signal conditioning, plus data acquisition, analysis, display, simulation and control capabilities.[1,2]

Visual programming tools are frequent] y used to simplify development (compared to text-based programming) of such systems. I imp] oyment of visual programming tools that control off-the-shelf interface cards has been the most important factor in reducing time and cost of configuring these systems. The MTC consistent] y achieves a reduction in software / systen~ development time by at least a factor of four, and up to an order of magnitude, compared to text-based software tools.[3,4,5,6] Others in industry are reporting similar increases in productivity and reduction in software / system development time and cost.[7,8,9]

## BACKGROUND

The Galileo spacecraft will arrive at Jupiter in December of 1995. It will be put into a highly elliptical orbit with a period of about three months. Each orbit will be modified slightly to allow the spacecraft to encounter a different moon or feature of Jupiter. For a few days during these close encounters, intense data acquisition will be performed with the data logged to the on-board tape recorder. During the remainder of each three-month orbit while the spacecraft is relatively far from Jupiter, the computer subsystems will be involved in compressing and compacting the tape data and downloading it to earth at a very low bit-rate (due to the high-rate dish antenna's failure to fully open).

Currently, the MTC is supporting a software redesign of the computer system aboard the Galileo spacecraft. This paper documents the programming effort to verify the correct re-programming of the Galileo computer subsystems by monitoring the telemetry of the ground Test Bed setup of the computer subsystems and the emulation hardware for the instruments to assure that every byte is correctly downloaded. The support is for the process, not the data itself. The MTC is using LabVIEW software among other tools to help test the flight software redesign. For details on the LabVIEW programming environment, other sources exist.[10,11,12]

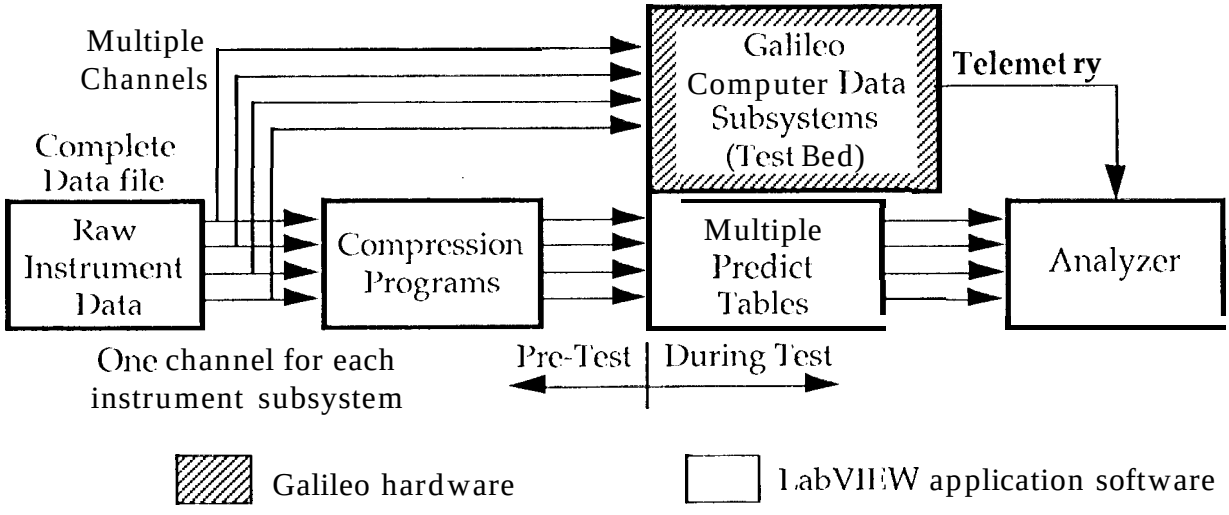


Figure 1. Ground Support Sequence

Figure 1 shows the Ground Support Sequence. The point of this effort is to test the compression algorithms plus the commutation of the data into packets. Using visual programming, software was developed to perform the various compression algorithms to be used on the different science instruments. Each instrument has multiple modes of compression to take into account the relative value of the data at differing times in the mission. The compressed data for each of these modes for each instrument are stored in files called the Predict Tables. A necessary additional component was a Test Bed simulator so all of the other programs could be developed and debugged before connection to the Test Bed.

During a test, the Test Bed reads the raw instrument data, performs the compression and commutation algorithms to be verified and outputs a telemetry stream. An analyzer was developed to monitor the telemetry from the Test Bed, decommutate the data, compare it to data in the Predict Tables and display the progress of the test.

LabVIEW running on a Macintosh Quadra was used as the programming environment for this task because it had proved to be superior in similar tasks.[13] The advantages LabVIEW provides include the ease with which the customer can communicate requirements to the programmers and understand the operation of the program so that changes can be suggested, The gains in productivity are attributed to the communication among the customer, developer, and computer that are facilitated by the visual syntax of the language. LabVIEW proved exceptionally capable in providing an integrated environment to manage all aspects of the telemetry test, from pre-test data set-up to post-test discrepancy resolution, as well as running the test in several simulator modes or with the Galileo hardware.

| | |
|---|---|
| HIC | Heavy Ion Counter Subsystem |
| EPD | Energetic Particles Detector Subsystem |
| PWL | Plasma Wave Subsystem (Low Rate) |
| PWH | Plasma Wave Subsystem (High Rate) |
| PLS | Plasma Subsystem |
| DDS | Dust Detector Subsystem |
| MAG | Magnetometer Subsystem |
| UVS | Ultraviolet Spectrometer Subsystem |
| EUV | Extreme Ultraviolet Subsystem |
| PPR | Photopolarimeter Radiometer Subsystem |
| NIMS | Near Infrared Mapping Spectrometer Subsystem |
| SSI | Solid State imaging Subsystem |
| OPN | Optical Navigation |
| AACS | Attitude and Articulation Control Subsystem |
| ENG | Engineering (housekeeping) |

Table 1. Galileo Instrument Subsystems

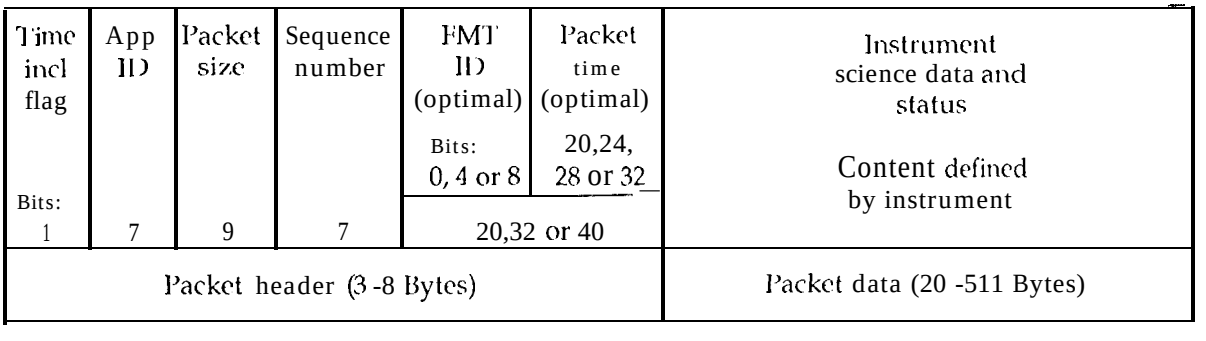


Figure 2. Packet structure

## ANALYZER

The telemetry from the Low-Gain Antenna Mission of the Galileo spacecraft contains data from the fifteen instrument sources. '1 'hey are assigned mnemonics as listed in l'able 1. Each of these instruments has from two to seven types of data or modes of operation which extend the mnemonic names with a single digit, There are a total of fifty-six of these instrument types and each is assigned an application identification (App **ID)** code. The data from these App ID's is independently collected into packets of up to 511 bytes and appended to a header of from three. to eight bytes (Figure 2). The packets are then assembled into VCDU's (Virtual Channel Data Units) which always contain four bytes of header and 442 bytes of packets with provisions for allowing packets to roll over from one VCDU to a later one (Figure 3). Four VCDU's (a total of 1784 bytes) arc then assembled into a frame with a two-byte frame number, an eight-byte PN (pseudo-noise) sync word, and 254 bytes of Reed-%lomon error-correction codes applied in eight unequal-size groups. This 2048-byte frame is then run through convolutional encoding which doubles the number of bytes producing 32768 bits of telemetry. Figure 4 is a schematic of the frame structure.

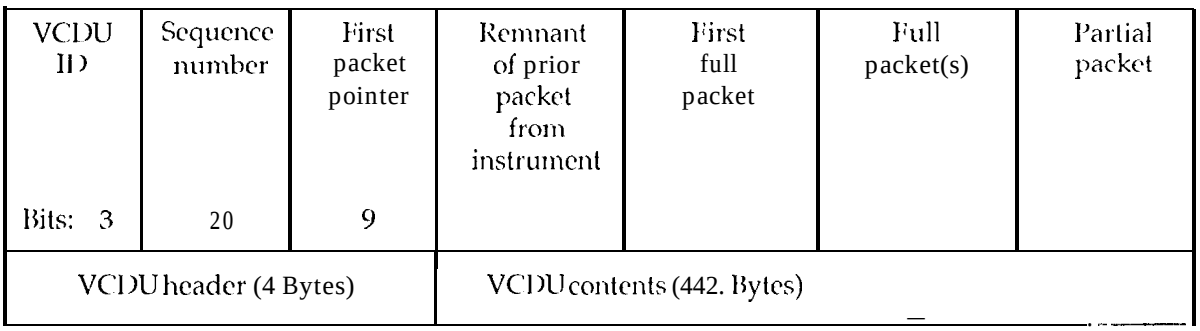


Figure 3. VCDU Structure

The testing of this telemetry stream involves a reverse process so that the data from the individual packets assigned to each of the App 1 I )'s can be recovered and compared to its predicted value. The first step is to capture the telemetry in real time. The telemetry stream from the Computer 1 )ata Subsystem consists of a clock line and a data line, both swinging between zero and five volts. The clock line is connected directly to the sample input on a data acquisition board. On each low-going transition of this clock line, the voltage on the data line is measured and stored in memory using the double-buffered data acquisition mode of LabVIEW which provides for continuous sampling of an input voltage. The sampled voltages are compared to a threshold (set at 2.5 volts) producing a single data bit for each clock.

The second step is to run these bits through a de-convolutional process which purposely does not correct for errors but produces two bit streams. The eight-byte PN sync word is searched for in both of these streams until it is found in one of them, at which point the other one is ignored. The next 2040 bytes are then assembled into a two-

byte frame sequence number, 254 bytes of Reed-Solomon error-correction codes, and four VCDU's o-f 446 bytes each.

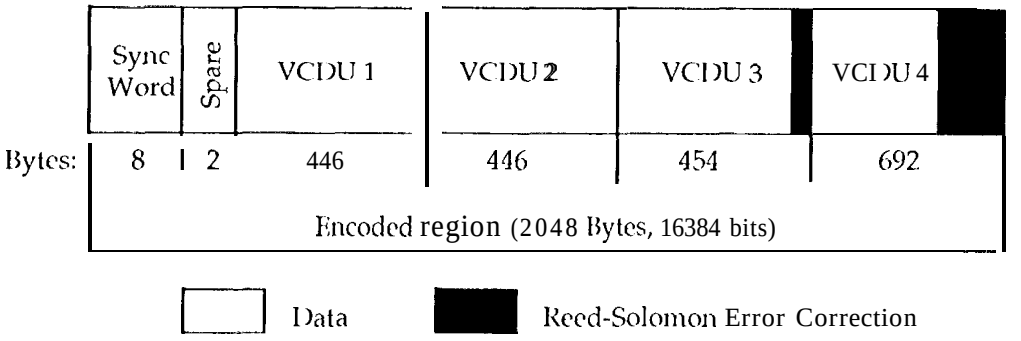


Figure 4. Telemetry Frame Structure

The full screen (21 inch monitor) user interface (LabVIEW Front Panel) is shown as Figure 5. It gives an idea of the complexity of the user interface required to display the analysis. It has been given boxes and" numbers to help explain in detail.

Figure .5 (part 1) displays the number of bits occurring before the sync word was found, the frame number and whether it is out of sequence, and whether the eight groups of Reed-Solomon codes arc incorrect (no corrections are applied). These (RS) errors are displayed in red in the frame in which they occur and change to yellow on subsequent frames. The operator can click on these latching error indicators, changing them to green. The buffer indicator displays the status of the real-time buffer. The Log TLM switch allows the operator to save the raw telemetry stream to a disk file for later analysis.

The third step is to check the headers of the four VCDU's (Figure 3). These headers contain three bits defining a VCDU ID type numbered zero through seven, twenty bits defining a sequence number, and nine bits used to handle the roll-over of packets between VCDU's. Each of the eight VCDU ID's keeps track of its own sequence number and can contain packets only from certain App ID's. Errors are again displayed in latching red, yellow, green indicators.

The fourth step (Figure .5, part 2) is to partition each VCDU into packets, temporarily storing any partial packet at the end and recombining any remnant packet at the beginning with its previously stored partial. The analyzer displays the sequence of packets within the four VCDU's contained in each frame in two different ways. First, a series of vertical text windows identifies information on each packet with three red-green error indicators below them. The top line of the text window displays the VCDU ID number followed by a letter signifying the position of the VCDU within the frame. Lower-case letters (a, b, c, & d) are used for partial packets at the end of the VCDU's and upper-case letters (A, B, C., & D) are used for complete packets and for remnant packets at the beginning of the VCDU's which have been combined with their

previously stored partials. The three error indicators below each packet text window are "turned off" (shown in gray) for the partial packets at the end of each VCDU because errors are not processed until the partial is combined with its remnant. The second line of the text window displays the App 1 D mnemonic and number. Further down the text window is the packet size which includes only those bytes within the current VCDU. The sum of the packet sizes for all the packets (including remnants and partials) within each VCDU will equal 442.

The second way that the packets within the four VCDU's in each frame are displayed is in a scrollable strip-chart (part 3). The frames are delineated with marks at the top and bottom of the strip-chart, The VCDU's are delineated with vertical gridlines, separating the frame into four parts. The first one (on the left) corresponds to the VCDU with the letter "A," the next one "B," then "C," and finally "D" on the right. The VCDU numbers are indicated by the colors of the stripes labeled "VCDU 1 I Y' on the strip-chart. White, for example, corresponds to VC1 )UID (J. The positions of the striped segments making up the lower two-thirds of the strip-chart indicate the App 11) mnemonics within each VCDU. Their colors indicate the App 1 I ) numbers and their lengths indicate their sizes. The errors are indicated by red stripes at the top of the strip-chart.

The fifth step is to display the packet header information in the text window (part 2). The current and previous sequence numbers are, displayed and the corresponding error indicator below the text window is turned red. Some App 11) types allow for a format **II** ) of four or eight bits which is used to interpret the data (Figure 2). These bits are displayed on the FMT ID line as one or two hex nybbles. The time of the packet (in spacecraft clock units) can be optionally included in the header. A "Time Included" bit in the header signifies whenever this happens. The actual number of bits of time varies depending on the App ID and is between 20 and 32 bits and is displayed as five to eight hex nybbles if present. When less than 32 bits, the more significant bits are discarded.

The sixth step is to analyze the packet data. The size is displayed on the I )ata Size line in the text window. Each App ID has associated with it a file containing the predicted telemetry bytes called the predict table. When each packet is received (including a remnant attached to a partial) it is searched for in its predict table. If it is found, its location is indicated in the text window at Prdt Tbl Pointer and the next expected location is saved in memory. The next time the same App 11) occurs, if the data in the packet is not found at the expected location, the Table Seq Error indicator will show red. If the data cannot be found in its predict table, the Not-In-Table Error indicator shows red. The three error indicators below each packet text window are not latched; i.e., they always show status for the current frame.

The seventh step is to update the small latched error indicators in the Packet Error Status Panel (part 4). Each App ID has a set of three indicators corresponding to the three indicators below the packet text windows, The one on the left is the Seq Error, the center one is the Table Seq Error, the right-hand one is the Not-In-Table Error. Any new errors during the current frame will appear as red and change to yellow on subsequent

frames. The operator can clear any of these indicators by clicking on them individually or all of them at once by hitting the Reset Errors button,

The eighth step is to update the large text window (part 5) which provides details on the errors. in addition to the information included in other places on the panel, offending packets are dumped so that post analysis can be performed. This text window can also be written to a file by turning on the Log Status switch.

The last step involves controlling other diagnostic windows under control of the operator, including packet windows which display all the data for a particular App ID, VCDU windows which display all the data for a particular VCDU type, a frame window which displays the entire unprocessed frame, and a statistics window which displays the current sequence number, the current format 1 D, and the latest included time for each App ID.

## CONCLUSIONS

A visual programming language was able to create, modify, test and display a telemetry stream. It provided easy visibility into the decommutation process modified by the Galileo programming support team. The time to write and modify the code using visual programming was significantly less (by a factor of 4 to 10) than using text-based code. This task showed that it is possible to use visual programming for realistic programming applications. It also confirmed that visual programming can significantly reduce software development time compared to text-based programming.

Other advantages demonstrated were in the areas of prototyping and verification. Different approaches can be demonstrated and evaluated quickly using a visual programming language. Verification can be demonstrated using the graphical user interface features available in a visual programming language easier than using conventional text-based code.

As stated, the gains in productivity are attributed to the communication among the customer, developer, and computer that arc facilitated by the visual syntax of the language. The advantages LabVIEW provides include the ease with which the customer can communicate requirements to the programmers and understand the operation of the program so that changes can be suggested. With this communication, the boundaries between requirements, design, development, and test appear to collapse.

## ACKNOWLEDGMENTS

The research described in this paper was carried out by the Jet Propulsion 1 laboratory, California institute of Technology, under a contract with the National Aeronautics and Space Administration.

## REFERENCES


1. Baroth, E. C., Clark, D. J. and Losey, R. W., "Acquisition, Analysis, Control, and Visualization of Data Using Personal Computers and a Graphical-Based Programming Language," Session 2659, Conference Proceedings of American Society of Engineering Educators (ASEE), Toledo, Ohio, June 21-25, 1992, pp. 1447-1453.
2. Baroth, E. C., Clark, D. J. and Losey, R. W., "An Adaptive Structure Data Acquisition System using a Graphical-Based Programming Language," AIAA-92-4833-CP, Conference Proceedings of Fourth AIAA/Air Force/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, Cleveland, Ohio, September 21-23, 1992, pp. 1104-1110.
3. Baroth, E. C., Hartsough, C., Johnsen, L., McGregor, J., Powell-Meeks, M., Walsh, A., Wells, G., Chazanoff, S., and Brunzie, T. "A Survey of Data Acquisition and Analysis Software Tools, Part 1," Evaluation Engineering Magazine, October, 1993, pp. 54-66.
4. Breeman, D., "Jet Propulsion Lab Aids in Space Craft Project," Scientific Computing and Automation, November, 1993, pp. 26-28.
5. Bulkeley, D., "Today's Equipment Tests Tomorrow's Designs," Design News Magazine, May 17,1993, pp. 82-86.
6. Puttre', M., "Software Makes Its Home in the Lab," -Mechanical Engineering Magazine, October, 1992, pp. 75-78.
7. Kent, G., "Automated RF Test System for Digital Cellular Telephones," Proceedings from NEPCON West '93, Anaheim, California, February 7-11, 1993, pp. 1055-1064.
8. Henderson, J. R., "Sequential File Creation for Automated Test Procedures," Proceedings from NEPCON West '93, Anaheim, California, February 7-11, 1993, pp. 1065-1077.
9. Jordan, S. C., "Cutting Costs the Old Fashioned Way," Proceedings from NEPCON West '93, Anaheim, California, February 7-11,1993, pp. 1921-1931.
10. National instruments Catalog, 1994, pp. 17-112.
11. Baroth, E. C., Hartsough, C., Johnsen, L., McGregor, J., Powell-Meeks, M., Walsh, A., Wells, G., Chazanoff, S., and Brunzie, T., "A Survey of Data Acquisition and Analysis Software Tools, Part 2," Evaluation Engineering Magazine, November, 1993, pp. 128-140.
12. Small, C. H., "Diagram Compilers Turn Pictures into Programs," EDN Magazine Special Software Supplement, June 20,1991, pp. 13-20.
13. Wells, G. and Baroth, E. C., "Telemetry Monitoring and Display using LabVIEW," Proceedings of National instruments User Symposium, Austin, Texas, March 28-30, 1993.

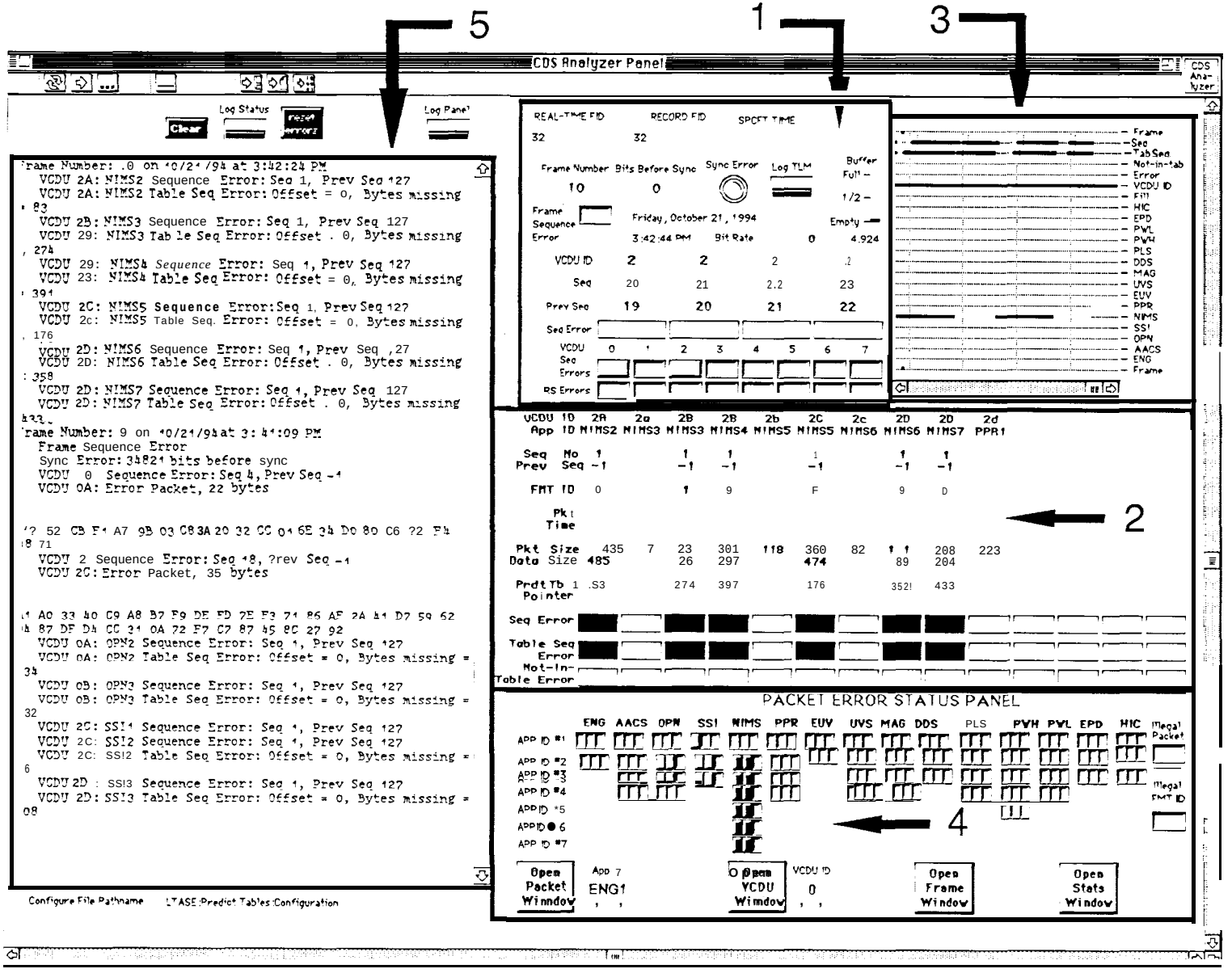


Figure 5. Analyzer User Interface (LabVIEW Front Panel)